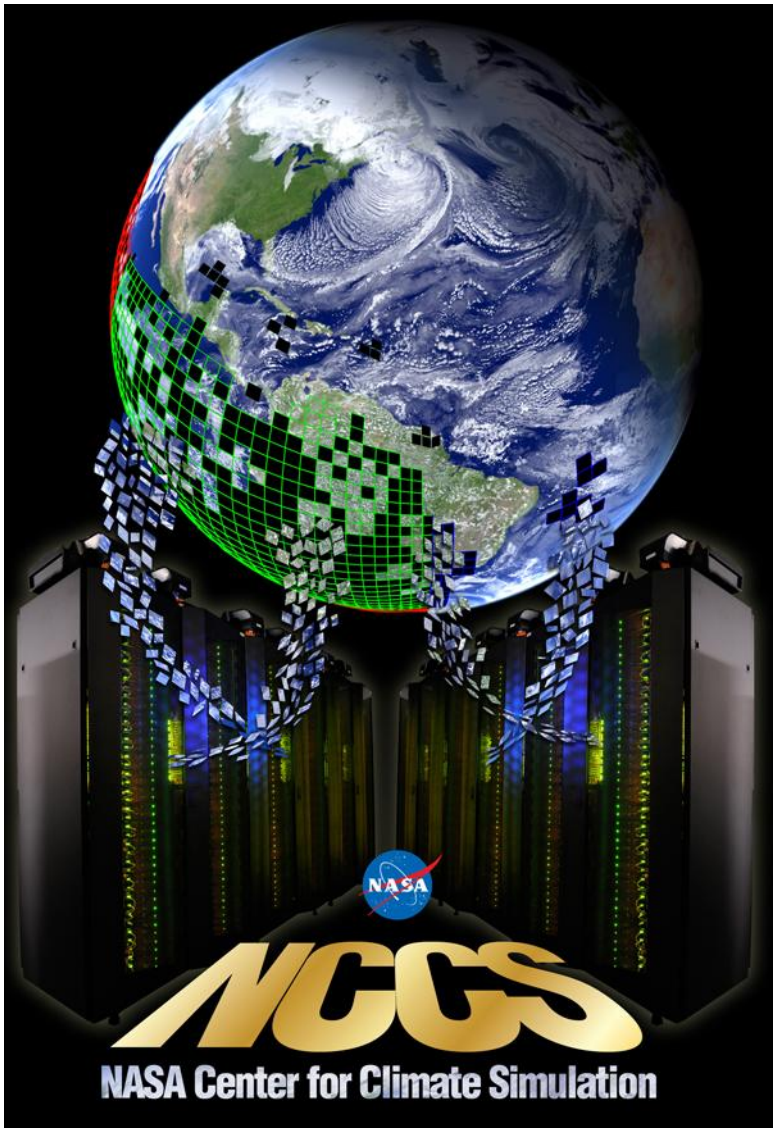
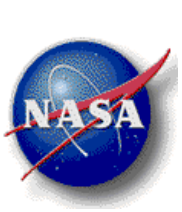


NCCS Brown Bag Series



Programming on the Intel MIC (Many Integrated Core) Architecture: An Introduction

Chongxun (Doris) Pan
doris.pan@nasa.gov
January 29, 2013

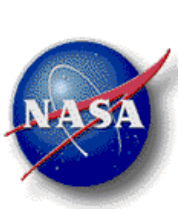
Agenda

- Discover SCU8 augmentation
- What is MIC?
- MIC Programming Considerations
- Offload *vs.* Native
- Demo

This is the first talk in the NCCS MIC tutorial series. Additional presentations will be upcoming on related topics.

Users can look at various documents and tutorial videos offered by Intel

http://software.intel.com/mic-developer



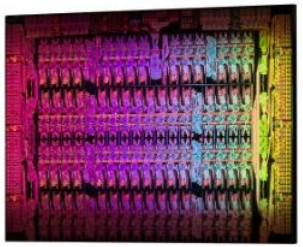
Some (confusing) Terminologies First

- **MIC**: Intel Many Integrated Core (MIC) architecture
- **Xeon**: Intel Xeon Processors, product codes named Nehalem, Westmere, Sandy Bridge (SNB), etc.
- **Xeon Phi**: Intel Xeon Phi coprocessors using the MIC architecture, the first product code name: Knights Corner (KNC)
- **Vectorization**: The process of transforming a scalar instruction that acts on single data element at a time (SISD), to a vector instruction that acts on multiple data elements at once (SIMD)

Discover SCU8

- Discover SCU8 augmentation includes
  - Intel Xeon Sandy Bridge processors: ~160 TFLOPS (480 nodes)
  - Intel Xeon Phi coprocessors:

  ~ 400 TFLOPS (480 KNC nodes)



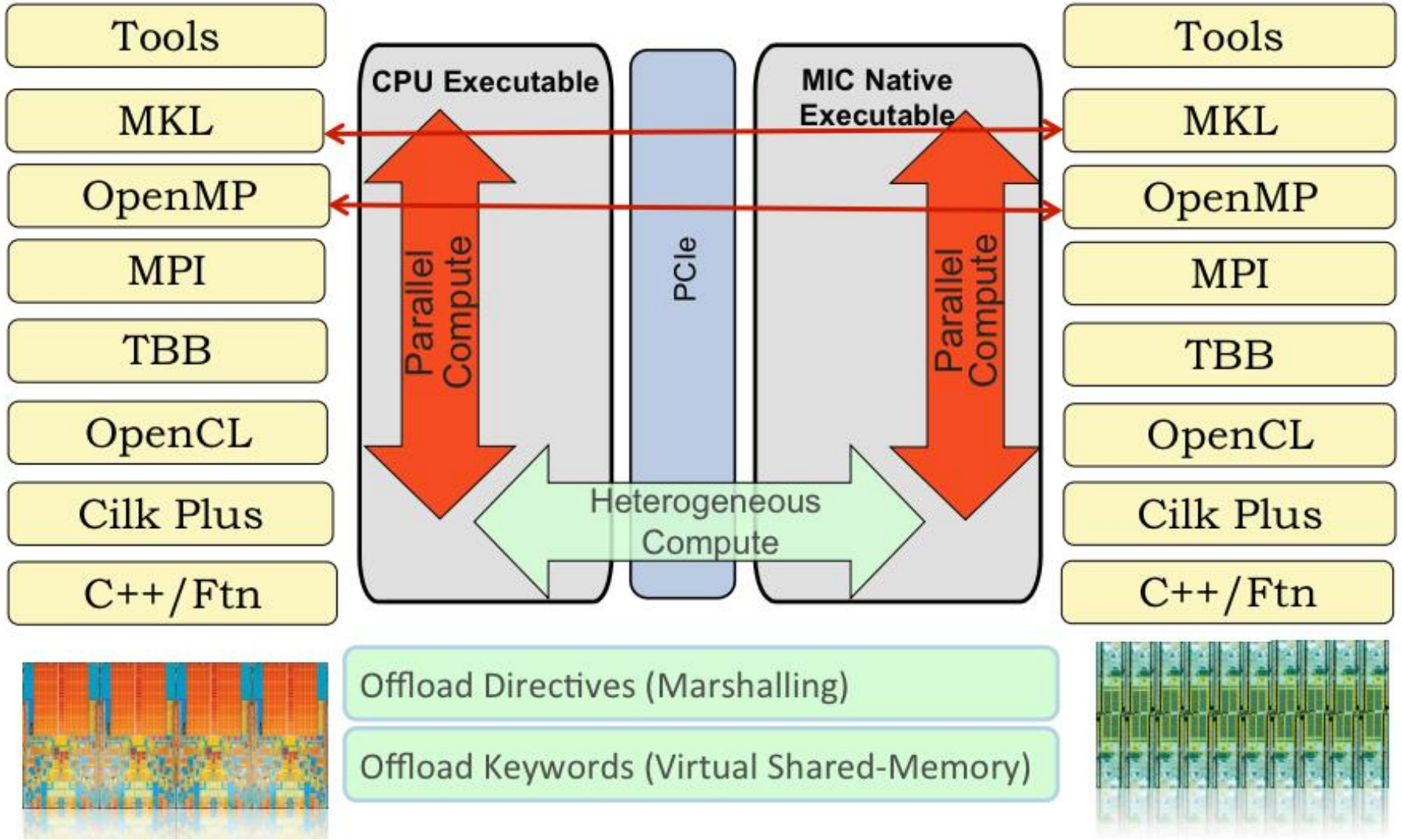
- SCU8 will be available to general users soon
  - Stage 1 (since Jan 23): SCU8 is open for limited user testing
  - Stage 2: SCU8 will be placed in a test queue for general user testing. Because the InfiniBand OFED software stack upgrade will be completed on SCU8 first, users will have to recompile their codes to run on SCU8
  - Stage 3: SCU8 in full production
  - Details/Time tables will be provided in announcements and the User Forum

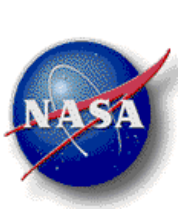
Intel MIC Overview

- Programming for Xeon Phi is similar to that for Xeon
  - Many slower x86 cores, but allow for more compute throughput
  - Any code can run on Xeon Phi coprocessors, not just kernels

- Specs for SCU8 KNC coprocessors
  - 60+ cores with 4 hardware threads/core on each coprocessor
  - Wide vector (SIMD) registers for more FP throughput
    - KNC - 512 bit vectors *vs.* SNB - 256 bit vectors
  - Cores interconnected by a high-speed bidirectional ring
  - Cores clocked at 1GHz
  - Coherent L1 and L2 caches
  - 512 KB L2 cache locally with high-speed access to all other caches
  - 8GB GDDR5 memory per coprocessor

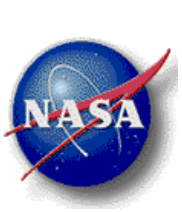
Intel MIC Overview – Software Architecture





Coprocessors *vs.* Accelerators -- Similarities

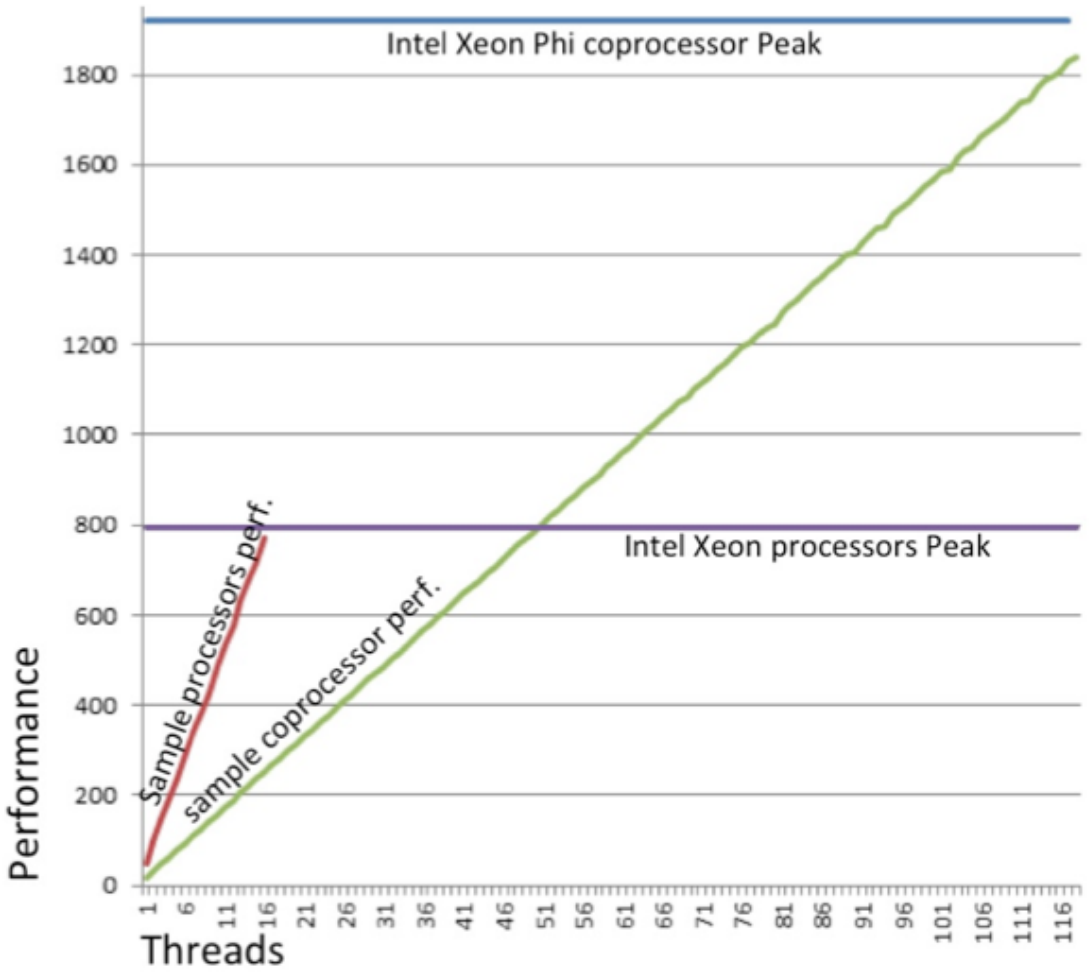
- Intel calls MIC “coprocessors”, and GPUs are often called “Accelerators”
- Similarities
  - Fast GDDR5 memory
  - Connected to the host through PCI-E bus. Two physical IP address spaces for the host and the MIC/GPU
  - Containing a large number of cores
- Applications that show positive performance with GPUs should always benefit with Xeon Phi because of the same fundamentals of vectorization or bandwidth.



Coprocessors *vs.* Accelerators -- Differences

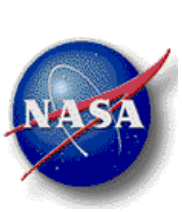
- Architecture:
  - x86 *vs.* streaming processors
  - Coherent caches *vs.* shared memory and caches
- HPC Programming model:
  - Extension to C/C++/Fortran. *vs.* CUDA / OpenCL
    OpenCL support upcoming
- MPI and Threading:
  - MPI and OpenMP *vs.* Hardware threads
  - MPI on host and on MIC *vs.* MPI on host only
- Programming details
  - Running natively or offloaded regions *vs.* Kernels
- Support for any code (serial/parallel) and scripting
  - Yes *vs.* No

Will my code run on the Xeon Phi coprocessors?



Well, very likely…

But, in getting an application ready to benefit from the Xeon Phi, nothing is more important than scaling of the application

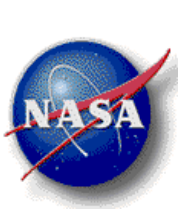


MIC Programming Considerations

- Xeon and Xeon Phi have different design goals. Xeon Phi is not intended to replace Xeon
  - It specializes in running **highly parallel and vectorized code**
  - Not optimized for processing serial code
- Not all applications can benefit from the capability of an Xeon Phi

- Very short tasks are not optimal for offload to the coprocessor
  - Costs that you need to amortize to make it worthwhile:
    - Overhead of code and data transfer to the coprocessor
    - Overhead of thread creation

- KNC comes with 8GB of memory total
  - So you trade off program space accessible to your code with offloading data transfer space



MIC Programming Considerations (*Cont'd*)

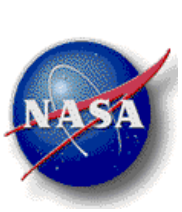
- Code porting is easy, although building some libraries can be a real pain
- Getting performance on MIC requires
  - Highly parallelized applications
    - Scaling to 100+ threads will be ideal
    - Making strong use of vector units
  - Programmer's effort on tuning and optimization
- "Optimize once, Run everywhere"
  - Tuning on Xeon Phi, for scaling, vectorization, and memory usage, will all benefit the application when running on the Xeon processors

How to involve Xeon Phi in an application?

	Host Only	Offload	Native (Phi only)	Native (Host and Phi)
Xeon (Host)	Program foo call bar() End	Program foo call bar() End	--	Program foo call bar End
Xeon Phi (Target)	--	bar ()	Program foo call bar End	Program foo call bar End

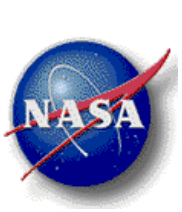
Heterogeneous (Offload) Model	Native Model
• Better serial processing • More memory • Better file access • Makes fuller use of available resources	• Almost no code change • More constraints in memory footprint and I/O • Maybe useful for quick testing

Currently Native model is not supported on SCU8 during the testing phases. In time it may change



Native *vs.* Offload

- An MPI program can be structured using either model
  - Processor-centric "offload" model where the program is viewed as running all MPI ranks on host processors and offloading select work to Phi
  - Native model with MPI ranks on both host and Phi
- A few Considerations using the Native model for MPI program
  - Fitting problems into the smaller memory on the Phi
  - Overhead of data transfers that favor minimization of communication to and from the Phi
  - Workload balancing between "big cores" on the host and "little cores" on the Phi
- An offload model can be attractive

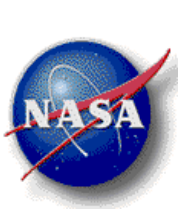


Offload

- The offload model for Xeon Phi is quite rich
  - Interoperable with OpenMP
  - Able to manage multiple coprocessor cards
  - Able to offload complex program components

- Two types of offload:
  1. Compiler Assisted offload (CAO):
     - Explicit control using offload pragmas/directives and keywords to make sections of code run on the Phi

  2. Automatically Offload (AO): Using some Intel MKL library routines



Compiler Assisted Offload (CAO)

- Similar to adding parallelism to serial code using OpenMP directives or Intel Cilk Plus keywords
  - Intel expects a future version of OpenMP will include offload directives
  - Again, **offload only the highly-parallel vectorized code**

- Heterogeneous Compiler
  - Builds a heterogeneous binary that runs on both the host and Phi
  - Adds code to transfer data automatically to the Phi and to start your code running (with no extra coding on your part)
  - The resulting binary runs whether or not a coprocessor is present

Offload Compiler/Execution Concept

CPU Program

```
f()
{
   #pragma offload
    target(mic)
      a = b + g();
}
```

```
__attribute__
((target(mic)))   g()
{
}
```

```
h()
{
}
```

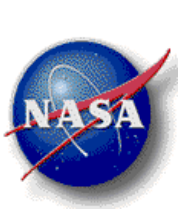
Contents of MIC Program

```
f_part1()
{
      a = b + g();
}
```

```
__attribute__
((target(mic)))   g()
{
}
```

Execution

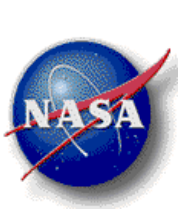
- At first offload, if Intel® MIC device is installed and available, MIC program is loaded
- At each offload, if Intel MIC device is present, statement is run on device, else statement runs on CPU
- At program termination, Intel MIC program is unloaded



Offload – Data Transfer

- Data has to be copied between host and Phi because they do not share a common memory
- Two techniques are available:

	Offload via Explicit Data Copying	**Offload via Implicit Data Copying**
Meaning …	Emulate shared data by copying back and forth at point of offload	Maintain coherence in a range of virtual addresses on host and Phi, automatically in software
Language Support	Fortran, C, C++	C, C++
Syntax	Pragmas/Directives: • `!dir$ [omp] offload` in Fortran • `#pragma offload` in C/C++	Keywords: `_Cilk_shared` and `_Cilk_offload`
Used for …	Offloads that transfer contiguous blocks of data	Offloads that transfer all or parts of complex data structures, or many small pieces of data



Fortran Syntax for Offload

- Compiler Directives
  - !DIR$ [OMP] OFFLOAD TARGET(MIC) …

  Offloads the following OpenMP block or function call
  - !DIR$ OFFLOAD BEGIN TARGET(MIC)…

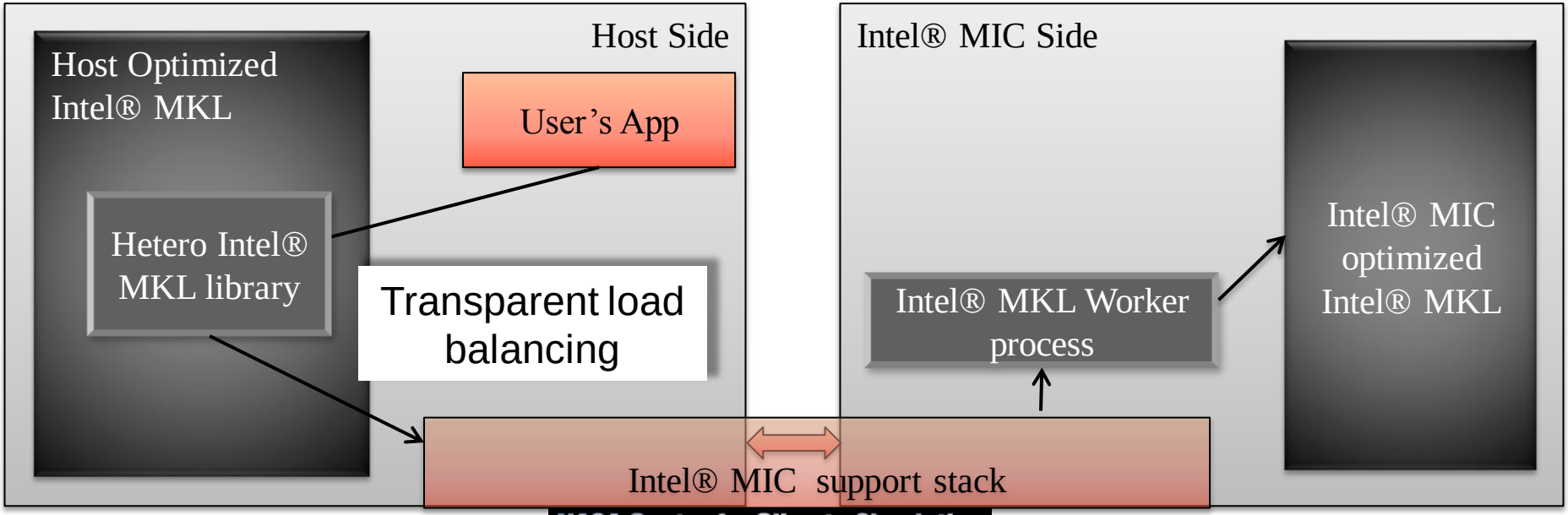
    !DIR$ END OFFLOAD

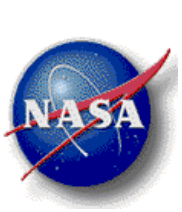
  Offloads a group of statements
- All procedures that will execute on the Phi must be declared as targeted for offload. Therefore,
  - In both the callee (required) and the caller (recommended), add:

  !DIR$ ATTRIBUTES OFFLOAD:MIC :: foo
- Offloaded data must be scalars, arrays or bit-wise copyable derived types (no embedded pointers or allocatable arrays)
  - Excludes most Fortran 2003 object-oriented constructs
  - Implicit copying is not supported in Fortran

Automatic Offload (AO) using Intel MKL

- For a selective set of AO-enabled routines (the list should expand in future MKL updates)
  - Offloading is automatic and transparent
  - MKL decides:
    - When to offload
    - Work division between host and targets
  - Users enjoy host and Phi parallelism automatically





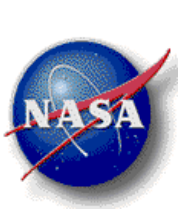
Automatic Offload (AO) using Intel MKL

- Easy to use:
  - Call a function “mkl_mic_enable” before calling MKL functions;

  or
  - Set an environmental variable

  setenv MKL_MIC_ENABLE 1
- When a MIC card is not connected, it just runs on the host as usual **without any penalty**
- You can control the workload after AO is enabled
  - call mkl_mic_set_workdivision(MKL_TARGET_HOST, 0, 0.5); or
  - setenv MKL_HOST_WORKDIVISION 50

- Compiled Assisted Offload can be used with MKL routines to provide more control
  - A big advantage is to reduce overhead by data persistence -- Reusing transferred data for multiple operations



CAO Example using Intel MKL routines

```
// Transfer matrices A, B, and C to coprocessor and do not de-allocate matrices A and B
#pragma offload target(mic) \
in(transa, transb, M, N, K, alpha, beta, LDA, LDB, LDC) \
in(A:length(NCOLA * LDA) free_if(0)) \
in(B:length(NCOLB * LDB) free_if(0)) \
inout(C:length(N * LDC))
{
sgemm(&transa, &transb, &M, &N, &K, &alpha, A, &LDA, B, &LDB, &beta, C, &LDC);
}
// Transfer matrix C1 to coprocessor and reuse matrices A and B
#pragma offload target(mic) \
in(transa1, transb1, M, N, K, alpha1, beta1, LDA, LDB, LDC1) \
nocopy(A:length(NCOLA * LDA) alloc_if(0) free_if(0)) \
nocopy(B:length(NCOLB * LDB) alloc_if(0) free_if(0)) \
inout(C1:length(N * LDC1))
{
sgemm(&transa1, &transb1, &M, &N, &K, &alpha1, A, &LDA, B, &LDB, &beta1, C1, &LDC1);
}
// Deallocate A and B on the coprocessor
#pragma offload target(mic) \
nocopy(A:length(NCOLA * LDA) free_if(1)) \
nocopy(B:length(NCOLB * LDB) free_if(1)) \ { }
```

0: false
1: true

Thread Controlling

- Avoid using the OS core of the Phi, which is for handling data transfer and housekeeping tasks

  Example: on a 60-core Phi coprocessor, max threads usable is 236.

setenv MIC_KMP_AFFINITY “explicit,granularity=fine, proclist=[1-236:1]”

- Also, set the thread affinity on the host

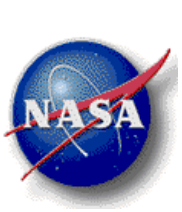
setenv KMP_AFFINITY “granularity=fine,compact”

- Different env-variables on host and Phi:

setenv MIC_ENV_PREFIX MIC

setenv OMP_NUM_THREADS 16

setenv MIC_OMP_NUM_THREADS 236



Thank You!

- Demo
  - micinfo/miccheck
  - Offload model. Compile and run a few programs

- More tutorials to come …
  - Intel MPI on MIC
  - Language Extensions for Offload
  - Maximize Vectorization
  - Performance analysis with VTune Amplifier
  - Performance tuning for MIC